# RKNN C API 参考手册

文件标识：RK-YH-YF-413

发布版本：V2.4.0

日期：2026-01-17

文件密级：□绝密 □秘密 □内部资料 ■公开

**免责声明**

本文档按"现状"提供，瑞芯微电子股份有限公司（"本公司"，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

**商标声明**

"Rockchip"、"瑞芯微"、"瑞芯"均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

**版权所有 © 2026 瑞芯微电子股份有限公司**

超越合理使用范畴，非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

---

**前言**

**概述**

本文是Rockchip RKNN C API 参考手册。

**读者对象**

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

**修订记录**

| 版本 | 修改人 | 修改日期 | 修改说明 | 核定人 |
| --- | --- | --- | --- | --- |
| v0.6.0 | HPC团队 | 2021-03-01 | 初始版本 | 熊伟 |
| v0.7.0 | HPC团队 | 2021-04-22 | 删除输入通道转换流程说明 | 熊伟 |
| v1.0.0 | HPC团队 | 2021-04-30 | 正式发布版本 | 熊伟 |
| v1.1.0 | HPC团队 | 2021-08-13 | 1. 增加rknn_tensor_mem_flags标志<br>2. 增加输入/输出tensor原生属性的查询命令<br>3.增加NC1HWC2的内存布局 | 熊伟 |
| v1.2.0b1 | NPU团队 | 2021-12-04 | 1. 增加RK3588/RK3588s平台说明<br>2. 增加rknn_set_core_mask接口<br>3. 增加rknn_dup_context接口<br>4. 增加输入输出API详细说明 | 熊伟 |
| v1.2.0 | HPC团队 | 2022-01-14 | 1. 增加关键字说明<br>2. 增加NPU SDK目录和编译说明<br>3. 增加调试方法章节<br>4. 增加NATIVE_LAYOUT中C2取值说明 | 熊伟 |
| v1.3.0 | NPU/HPC团队 | 2022-05-13 | 1. 修复命名destroy变为destroy<br>2. 增加RV1106/RV1103的使用说明<br>3. 增加NATIVE_LAYOUT的细节说明<br>4. 增加C API硬件平台支持说明<br>5. 增加NPU版本、利用率查询以及NPU电源手动开关的指令 | 熊伟 |
| v1.4.0 | NPU/HPC团队 | 2022-08-31 | 1. RV1106/RV1103增加rknn_create_mem_from_phys/ rknn_create_mem_from_fd/ rknn_set_weight_mem/ rknn_set_internal_mem接口支持<br>2. 新增权重共享的功能<br>3. RK3588新增sram功能支持<br>4. RK3588新增单batch多核支持<br>5. NPU新版本驱动增加查询频率、电压、设置延时关闭时间等功能 | 熊伟 |
| v1.4.2 | HPC团队 | 2023-02-13 | 1. 增加RK3562的使用说明<br>2. 增加rknn_init接口中RKNN_FLAG_COLLECT_MODEL_INFO_ONLY标志说明 | 熊伟 |
| v1.5.0 | HPC团队 | 2023-05-22 | 1. 增加动态形状输入API使用说明和相关数据结构说明<br>2. 增加Matmul API使用说明 | 熊伟 |
| v1.5.2 | HPC团队 | 2023-08-22 | 1. 增加rknn_init接口中RKNN_FLAG_EXECUTE_FALLBACK_PRIOR_DEVICE_GPU和RKNN_FLAG_INTERNAL_ALLOC_OUTSIDE标志说明<br>2. 移除旧动态输入形状功能的rknn_set_input_shape接口说明，新增rknn_set_input_shapes接口说明 | 熊伟 |
| v1.6.0 | HPC团队 | 2023-11-28 | 1. 增加rknn_init接口中RKNN_FLAG_ENABLE_SRAM、RKNN_FLAG_SHARE_SRAM以及RKNN_FLAG_DISABLE_PROC_HIGH_PRIORITY标志说明<br>2. 增加rknn_set_batch_core_num接口说明<br>3. 增加rknn_mem_sync接口说明 | 熊伟 |

| 版本 | 修改人 | 修改日期 | 修改说明 | 核定人 |
| --- | --- | --- | --- | --- |
| v2.0.0-beta0 | HPC团队 | 2024-03-18 | 1. 增加rknn_create_mem2接口说明<br>2. 修改rknn_matmul_info和rknn_matmul_type结构体<br>3. 增加rknn_quant_params和rknn_matmul_shape结构体<br>4. 增加rknn_matmul_set_quant_params, rknn_matmul_get_quant_params, rknn_matmul_create_dyn_shape, rknn_matmul_set_dynamic_shape接口 | 熊伟 |
| v2.1.0 | HPC团队 | 2024-08-02 | 1. 增加RV1103B和RK2118平台说明<br>2. 增加RKNN_FLAG_DISABLE_FLUSH_INPUT_MEM_CACHE，RKNN_FLAG_DISABLE_FLUSH_OUTPUT_MEM_CACHE，RKNN_FLAG_MODEL_BUFFER_ZERO_COPY初始化标志说明<br>3. 增加RKNN_MEM_FLAG_ALLOC_NO_CONTEXT内存分配标志说明<br>4. 增加RKNN_QUERY_DEVICE_MEM_INFO查询命令说明<br>5. 修改rknn_init_extend、rknn_matmul_info、rknn_matmul_type和rknn_quant_params结构体<br>6. 修改rknn_B_normal_layout_to_native_layout接口<br>7. 增加RKNN_FLOAT16_MM_INT4_TO_FLOAT32，RKNN_FLOAT16_MM_INT4_TO_FLOAT16，RKNN_INT8_MM_INT4_TO_INT32类型支持<br>8. 增加rknn_matmul_layout数据结构，增加B_layout=RKNN_MM_LAYOUT_TP_NORM支持<br>9. 增加rknn_matmul_quant_type数据结构，支持Matmul API的B矩阵分组对称量化<br>10. 废弃rknn_matmul_create_dyn_shape,改用rknn_matmul_create_dynamic_shape接口 | 熊伟 |
| v2.2.0 | HPC团队 | 2024-09-04 | 更新版本号 | 熊伟 |
| v2.3.0 | HPC团队 | 2024-11-06 | 更新版本号 | 熊伟 |
| v2.3.2 | HPC团队 | 2025-04-03 | 增加 RV1126B 平台支持 | 熊伟 |
| v2.4.0 | HPC团队 | 2026-01-17 | 更新版本号 | 熊伟 |

## 目录


**RKNN C API 参考手册**

1. 概述
2. 硬件平台
3. RKNPU编译说明
    3.1 RKNN C API头文件
    3.2 Linux平台RKNPU运行时库
    3.3 Android平台RKNPU运行时库
    3.4 RK2118平台RKNPU运行时库
4. RKNN C API说明
    4.1 各个硬件平台的C API支持情况
    4.2 基础数据结构定义
        rknn_sdk_version
        rknn_input_output_num
        rknn_input_range
        rknn_tensor_attr
        rknn_perf_detail
        rknn_perf_run
        rknn_mem_size
        rknn_tensor_mem
        rknn_input
        rknn_output
        rknn_init_extend
        rknn_run_extend
        rknn_output_extend
        rknn_custom_string
    4.3 基础API说明
        rknn_init
        rknn_set_core_mask
        rknn_set_batch_core_num
        rknn_dup_context
        rknn_destroy
        rknn_query
        rknn_inputs_set
        rknn_run
        rknn_outputs_get
        rknn_outputs_release
        rknn_create_mem_from_phys
        rknn_create_mem_from_fd
        rknn_create_mem
        rknn_create_mem2
        rknn_destroy_mem
        rknn_set_weight_mem
        rknn_set_internal_mem
        rknn_set_io_mem
        rknn_set_input_shape（deprecated）
        rknn_set_input_shapes
        rknn_mem_sync
    4.4 矩阵乘法数据结构定义
        rknn_matmul_info
        rknn_matmul_tensor_attr

rknn_matmul_io_attr
rknn_quant_params
rknn_matmul_shape
4.5 矩阵乘法API说明
rknn_matmul_create
rknn_matmul_set_io_mem
rknn_matmul_set_core_mask
rknn_matmul_set_quant_params
rknn_matmul_get_quant_params
rknn_matmul_create_dyn_shape(deprecated)
rknn_matmul_create_dynamic_shape
rknn_matmul_set_dynamic_shape
rknn_B_normal_layout_to_native_layout
rknn_matmul_run
rknn_matmul_destroy
4.6 自定义算子数据结构定义
rknn_gpu_op_context
rknn_custom_op_context
rknn_custom_op_tensor
rknn_custom_op_attr
rknn_custom_op
4.7 自定义算子API说明
rknn_register_custom_ops
rknn_custom_op_get_op_attr
5. RKNN返回值错误码

# 1. 概述

RKNN C API是RKNPU Runtime（运行时库）的C语言接口。通过使用RKNN C API，开发者可以利用NPU的计算能力完成高效的RKNN模型推理或矩阵乘法计算任务。本文对RKNN C API的各个函数、数据结构以及返回码定义进行说明。

# 2. 硬件平台

本文档适用如下硬件平台：

- RV1103
- RV1103B
- RV1106
- RV1106B
- RV1126B
- RK2118
- RK3562
- RK3566系列
- RK3568系列
- RK3576系列
- RK3588系列

# 3. RKNPU编译说明

开发者编译应用时要包含接口函数所在的头文件，并且根据使用的硬件平台和系统类型，链接相应RKNPU运行时库。以下对RKNN C API头文件和运行时库文件进行说明。

## 3.1 RKNN C API头文件

根据不同的功能特点，RKNN C API的接口分为三个部分，各个部分函数、数据结构定义和头文件对应关系如下：

1. “rknn_api.h”定义部署RKNN模型的基础接口和数据结构。
2. “rknn_matmul_api.h”定义矩阵乘法接口和数据结构。
3. “rknn_custom_op.h”定义用户自定义算子接口和数据结构。

## 3.2 Linux平台RKNPU运行时库

1. 对于RK3562、RK3566系列、RK3568系列、RK3576系列、RK3588系列、RV1126B硬件平台，RKNPU运行时库文件为<sdk_path>/rknpu2/runtime目录下的librknnrt.so，其中<sdk_path>是瑞芯微NPU软件开发包的路径。
2. 对于RV1106、RV1103、RV1106B、RV1103B硬件平台，RKNPU运行时库文件为<sdk_path>/rknpu2/runtime目录下的librknnmrt.so。

## 3.3 Android平台RKNPU运行时库

Android平台有两种方式来调用RKNN C API：

1. 应用直接链接librknnrt.so。
2. 应用链接Android平台HIDL实现的librknn_api_android.so。

对于需要通过CTS/VTS测试的Android设备需要使用基于Android平台HIDL实现的RKNN API（**librknn_api_android.so不包含矩阵乘法和自定义算子功能**）。如果不需要通过CTS/VTS测试的设备建议直接使用librknnrt.so（包含矩阵乘法和自定义算子功能），对各个接口调用流程的链路更短，可以提供更好的性能。

对于使用Android HIDL实现的RKNN API的代码位于RK3562/RK3566/RK3568/RK3576/RK3588 Android系统SDK的vendor/rockchip/hardware/interfaces/neuralnetworks目录下。当完成Android系统编译后，将会生成RKNPU相关的一系列库文件（对于应用开发只需要链接使用librknn_api_android.so即可），如下所示：

```
/vendor/lib/librknn_api_android.so
/vendor/lib/librknnhal_bridge.rockchip.so
/vendor/lib64/librknn_api_android.so
/vendor/lib64/librknnhal_bridge.rockchip.so
/vendor/lib64/rockchip.hardware.neuralnetworks@1.0.so
/vendor/lib64/rockchip.hardware.neuralnetworks@1.0-adapter-helper.so
/vendor/lib64/hw/rockchip.hardware.neuralnetworks@1.0-impl.so
/vendor/bin/hw/rockchip.hardware.neuralnetworks@1.0-service
```

也可以使用如下命令单独重新编译生成以上的库文件：

```
mmm vendor/rockchip/hardware/interfaces/neuralnetworks/ -j8
```

## 3.4 RK2118平台RKNPU运行时库

对于RK2118硬件平台，RKNPU运行时库文件位于RK2118的系统SDK中，如何获取请参考《**Rockchip_RK2118_Quick_Start_RKNN_SDK_CN.pdf**》的"准备RK2118 SDK"章节。

# 4. RKNN C API说明

## 4.1 各个硬件平台的C API支持情况

由于不同的芯片平台的硬件特性不同，RKNN C API的接口以及接口参数的支持情况也不同。各个硬件平台的RKNN C API接口支持情况如表4-1所示：

表4-1 各个硬件平台的RKNN C API接口支持情况

| | RKNN C API | RV1126B/RK3562/RK3566/RK3568 | RK3588/RK3576 | RV1106/RV1103/RV1103B/RV1106B | RK2118 |
|---|---|---|---|---|---|
| 1 | rknn_init | √ | √ | √ | √ |
| 2 | rknn_set_core_mask | × | √ | × | × |
| 3 | rknn_dup_context | √ | √ | × | × |
| 4 | rknn_destroy | √ | √ | √ | √ |
| 5 | rknn_query | √ | √ | √ | √ |
| 6 | rknn_inputs_set | √ | √ | × | × |
| 7 | rknn_run | √ | √ | √ | √ |
| 8 | rknn_wait | × | × | × | × |
| 9 | rknn_outputs_get | √ | √ | × | × |
| 10 | rknn_outputs_release | √ | √ | × | × |
| 11 | rknn_create_mem_from_mb_blk | × | × | × | × |
| 12 | rknn_create_mem_from_phys | √ | √ | √ | √ |
| 13 | rknn_create_mem_from_fd | √ | √ | √ | × |
| 14 | rknn_create_mem | √ | √ | √ | √ |
| 15 | rknn_destroy_mem | √ | √ | √ | √ |
| 16 | rknn_set_weight_mem | √ | √ | √ | √ |
| 17 | rknn_set_internal_mem | √ | √ | √ | √ |
| 18 | rknn_set_io_mem | √ | √ | √ | √ |
| 19 | rknn_set_input_shapes | √ | √ | × | × |
| 20 | rknn_mem_sync | √ | √ | √ | √ |
| 21 | rknn_matmul_create | √ | √ | × | × |
| 22 | rknn_matmul_set_io_mem | √ | √ | × | × |
| 23 | rknn_matmul_set_core_mask | × | √ | × | × |
| 24 | rknn_matmul_run | √ | √ | × | × |
| 25 | rknn_matmul_destroy | √ | √ | × | × |
| 26 | rknn_register_custom_ops | √ | √ | × | × |
| 27 | rknn_custom_op_get_op_attr | √ | √ | × | × |
| 28 | rknn_set_batch_core_num | × | √ | × | × |
| 29 | rknn_matmul_set_quant_params | √ | √ | × | × |
| 30 | rknn_matmul_get_quant_params | √ | √ | × | × |
| 31 | rknn_matmul_create_dynamic_shape | √ | √ | × | × |
| 32 | rknn_matmul_set_dynamic_shape | √ | √ | × | × |
| 33 | rknn_B_normal_layout_to_native_layout | √ | √ | × | × |

| | RKNN C API | RV1126B/RK3562/RK3566/ RK3568 | RK3588/RK3576 | RV1106/RV1103/ RV1103B/RV1106B | RK2118 |
|---|---|---|---|---|---|
| 34 | rknn_create_mem2 | √ | √ | √ | √ |

各个硬件平台使用rknn_query函数支持的查询参数如表4-2所示：

表4-2 各个硬件平台rknn_query函数支持的查询参数

| | rknn_query参数 | RV1126B/RK3562/RK3566/ RK3568 | RK3576/ RK3588 | RV1106/RV1106B/ RV1103/RV1103B/RK2118 |
|---|---|---|---|---|
| 1 | RKNN_QUERY_IN_OUT_NUM | √ | √ | √ |
| 2 | RKNN_QUERY_INPUT_ATTR | √ | √ | √ |
| 3 | RKNN_QUERY_OUTPUT_ATTR | √ | √ | √ |
| 4 | RKNN_QUERY_PERF_DETAIL | √ | √ | × |
| 5 | RKNN_QUERY_PERF_RUN | √ | √ | × |
| 6 | RKNN_QUERY_SDK_VERSION | √ | √ | √ |
| 7 | RKNN_QUERY_MEM_SIZE | √ | √ | √ |
| 8 | RKNN_QUERY_CUSTOM_STRING | √ | √ | √ |
| 9 | RKNN_QUERY_NATIVE_INPUT_ATTR | √ | √ | √ |
| 10 | RKNN_QUERY_NATIVE_OUTPUT_ATTR | √ | √ | √ |
| 11 | RKNN_QUERY_NATIVE_NC1HWC2_INPUT_ATTR | √ | √ | √ |
| 12 | RKNN_QUERY_NATIVE_NC1HWC2_OUTPUT_ATTR | √ | √ | √ |
| 13 | RKNN_QUERY_NATIVE_NHWC_INPUT_ATTR | √ | √ | √ |
| 14 | RKNN_QUERY_NATIVE_NHWC_OUTPUT_ATTR | √ | √ | √ |
| 15 | RKNN_QUERY_INPUT_DYNAMIC_RANGE | √ | √ | × |
| 16 | RKNN_QUERY_DEVICE_MEM_INFO | × | × | √ |
| 17 | RKNN_QUERY_CURRENT_INPUT_ATTR | √ | √ | × |
| 18 | RKNN_QUERY_CURRENT_OUTPUT_ATTR | √ | √ | × |
| 19 | RKNN_QUERY_CURRENT_NATIVE_INPUT_ATTR | √ | √ | × |
| 20 | RKNN_QUERY_CURRENT_NATIVE_OUTPUT_ATTR | √ | √ | × |

## 4.2 基础数据结构定义

### rknn_sdk_version

结构体rknn_sdk_version用来表示RKNN SDK的版本信息，结构体的定义如下：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| api_version | char[] | SDK的版本信息。 |
| drv_version | char[] | SDK所基于的驱动版本信息。 |

### rknn_input_output_num

结构体rknn_input_output_num表示输入输出tensor个数，其结构体成员变量如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| n_input | uint32_t | 输入tensor个数。 |
| n_output | uint32_t | 输出tensor个数。 |

### rknn_input_range

结构体rknn_input_range表示一个输入的支持形状列表信息。它包含了输入的索引、支持的形状个数、数据布局格式、名称以及形状列表，具体的结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| index | uint32_t | 表示该形状对应输入的索引位置。 |
| shape_number | uint32_t | 表示RKNN模型支持的输入形状个数。 |
| fmt | rknn_tensor_format | 表示形状对应的数据布局格式。 |
| name | char[] | 表示输入的名称。 |
| dyn_range | uint32_t[][] | 表示输入形状列表，它是包含多个形状数组的二维数组，形状优先存储。 |
| n_dims | uint32_t | 表示每个形状数组的有效维度个数。 |

## rknn_tensor_attr

结构体rknn_tensor_attr表示模型的tensor的属性，结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| index | uint32_t | 表示输入输出tensor的索引位置。 |
| n_dims | uint32_t | Tensor维度个数。 |
| dims | uint32_t[] | Tensor形状。 |
| name | char[] | Tensor名称。 |
| n_elems | uint32_t | Tensor数据元素个数。 |
| size | uint32_t | Tensor数据所占内存大小。 |
| fmt | rknn_tensor_format | Tensor维度的格式，有以下格式： **RKNN_TENSOR_NCHW**, **RKNN_TENSOR_NHWC**, **RKNN_TENSOR_NC1HWC2**, **RKNN_TENSOR_UNDEFINED** |
| type | rknn_tensor_type | Tensor数据类型，有以下数据类型： **RKNN_TENSOR_FLOAT32**, **RKNN_TENSOR_FLOAT16**, **RKNN_TENSOR_INT8**, **RKNN_TENSOR_UINT8**, **RKNN_TENSOR_INT16**, **RKNN_TENSOR_UINT16**, **RKNN_TENSOR_INT32**, **RKNN_TENSOR_INT64**, **RKNN_TENSOR_BOOL** |
| qnt_type | rknn_tensor_qnt_type | Tensor量化类型，有以下的量化类型：<br>**RKNN_TENSOR_QNT_NONE**：未量化；<br>**RKNN_TENSOR_QNT_DFP**：动态定点量化；<br>**RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC**：非对称量化。 |
| fl | int8_t | RKNN_TENSOR_QNT_DFP量化类型的参数。 |
| scale | float | RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC量化类型的参数。 |
| w_stride | uint32_t | 实际存储一行图像数据的像素数目，等于一行的有效数据像素数目 + 为硬件快速跨越到下一行而补齐的一些无效像素数目，单位是像素。 |
| size_with_stride | uint32_t | 实际存储图像数据所占的存储空间的大小（包括了补齐的无效像素的存储空间大小）。 |
| pass_through | uint8_t | 0表示未转换的数据，1表示转换后的数据，转换包括归一化和量化。 |
| h_stride | uint32_t | 仅用于多batch输入场景，且**该值由用户设置**。目的是NPU正确地读取每batch数据的起始地址，它等于原始模型的输入高度+跨越下一列而补齐的无效像素数目。如果设置成0，表示与原始模型输入高度一致，单位是像素。 |

## rknn_perf_detail

结构体rknn_perf_detail表示模型的性能详情，结构体的定义如下表所示
（RV1106/RV1106B/RV1103/RV1103B/RK2118暂不支持）：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| perf_data | char* | 性能详情包含网络每层运行时间，能够直接打印出来查看。 |
| data_len | uint64_t | 存放性能详情的字符串数组的长度。 |

## rknn_perf_run

结构体rknn_perf_run表示模型的总体性能，结构体的定义如下表所示
（RV1106/RV1106B/RV1103/RV1103B/RK2118暂不支持）：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| run_duration | int64_t | 网络总体运行（不包含设置输入/输出）时间，单位是微秒。 |

## rknn_mem_size

结构体rknn_mem_size表示初始化模型时的内存分配情况，结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| total_weight_size | uint32_t | 模型的权重占用的内存大小。 |
| total_internal_size | uint32_t | 模型的中间tensor占用的内存大小。 |
| total_dma_allocated_size | uint64_t | 模型申请的所有dma内存之和。 |
| total_sram_size | uint32_t | 只针对RK3588有效，为NPU预留的系统SRAM大小（具体使用方式参考 《RK3588_NPU_SRAM_usage.md》 ）。 |
| free_sram_size | uint32_t | 只针对RK3588有效，当前可用的空闲SRAM大小（具体使用方式参考《RK3588_NPU_SRAM_usage.md》）。 |
| reserved[12] | uint32_t | 预留。 |

## rknn_tensor_mem

结构体rknn_tensor_mem表示tensor的内存信息。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| virt_addr | void* | 该tensor的虚拟地址。 |
| phys_addr | uint64_t | 该tensor的物理地址。 |
| fd | int32_t | 该tensor的文件描述符。 |
| offset | int32_t | 相较于文件描述符和虚拟地址的偏移量。 |
| size | uint32_t | 该tensor占用的内存大小。 |
| flags | uint32_t | rknn_tensor_mem的标志位，有以下标志：<br>**RKNN_TENSOR_MEMORY_FALGS_ALLOC_INSIDE:** 表明rknn_tensor_mem结构体由运行时创建；<br>**RKNN_TENSOR_MEMORY_FLAGS_FROM_FD:** 表明rknn_tensor_mem结构体由fd构造；<br>**RKNN_TENSOR_MEMORY_FLAGS_FROM_PHYS:** 表明rknn_tensor_mem结构体由物理地址构造； **用户不用关注该标志**。 |
| priv_data | void* | 内存的私有数据。 |

## rknn_input

结构体rknn_input表示模型的一个数据输入，用来作为参数传入给rknn_inputs_set函数。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| index | uint32_t | 该输入的索引位置。 |
| buf | void* | 输入数据的指针。 |
| size | uint32_t | 输入数据所占内存大小。 |
| pass_through | uint8_t | 设置为1时会将buf存放的输入数据直接设置给模型的输入节点，不做任何预处理。 |
| type | rknn_tensor_type | 输入数据的类型。 |
| fmt | rknn_tensor_format | 输入数据的格式。 |

## rknn_output

结构体rknn_output表示模型的一个数据输出，用来作为参数传入给rknn_outputs_get函数，在函数执行后，结构体对象将会被赋值。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| want_float | uint8_t | 标识是否需要将输出数据转为float类型输出，该字段由用户设置。 |
| is_prealloc | uint8_t | 标识存放输出数据是否是预分配，该字段由用户设置。 |
| index | uint32_t | 该输出的索引位置，该字段由用户设置。 |
| buf | void* | 输出数据的指针，该字段由接口返回。 |
| size | uint32_t | 输出数据所占内存大小，该字段由接口返回。 |

## rknn_init_extend

结构体rknn_init_extend表示初始化模型时的扩展信息。结构体的定义如下表所示（RV1106/RV1106B/RV1103/RV1103B/RK2118暂不支持）：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| ctx | rknn_context | 已初始化的rknn_context对象。 |
| real_model_offset | int32_t | 真正rknn模型在文件中的偏移，只有以文件路径为参数或零拷贝模型内存初始化时才生效。 |
| real_model_size | uint32_t | 真正rknn模型在文件中的大小，只有以文件路径为参数或另拷贝模型内存初始化时才生效。 |
| model_buffer_fd | int32_t | 使用RKNN_FLAG_MODEL_BUFFER_ZERO_COPY标志初始化后，NPU分配的模型内存代表的fd。 |
| model_buffer_flags | uint32_t | 使用RKNN_FLAG_MODEL_BUFFER_ZERO_COPY标志初始化后，NPU分配的模型内存代表的内存标志。 |
| reserved | uint8_t[] | 预留数据位。 |

## rknn_run_extend

结构体rknn_run_extend表示模型推理时的扩展信息，**目前暂不支持使用**。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| frame_id | uint64_t | 表示当前推理的帧序号。 |
| non_block | int32_t | 0表示阻塞模式，1表示非阻塞模式，非阻塞即rknn_run调用直接返回。 |
| timeout_ms | int32_t | 推理超时的上限，单位毫秒。 |
| fence_fd | int32_t | 用于非阻塞执行推理，**暂不支持**。 |

## rknn_output_extend

结构体rknn_output_extend表示获取输出的扩展信息，**目前暂不支持使用**。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| frame_id | int32_t | 输出结果的帧序号。 |

## rknn_custom_string

结构体rknn_custom_string表示转换RKNN模型时，用户设置的自定义字符串，结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| string | char[] | 用户自定义字符串。 |

## 4.3 基础API说明

### rknn_init

rknn_init初始化函数功能为创建rknn_context对象、加载RKNN模型以及根据flag和rknn_init_extend结构体执行特定的初始化行为。

| API | rknn_init |
| --- | --- |
| 功能 | 初始化rknn上下文。 |
| 参数 | rknn_context *context：rknn_context指针。 |
| | void *model：RKNN模型的二进制数据或者RKNN模型路径。当参数size大于0时，model表示二进制数据；当参数size等于0时，model表示RKNN模型路径。 |
| | uint32_t size：当model是二进制数据，表示模型大小，当model是路径，则设置为0。 |
| | uint32_t flag：初始化标志，默认初始化行为需要设置为0。 |
| | rknn_init_extend：特定初始化时的扩展信息。没有使用，传入NULL即可。如果需要共享模型weight内存，则需要传入另个模型rknn_context指针。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0, NULL);
```

各个初始化标志说明如下：

**RKNN_FLAG_COLLECT_PERF_MASK**：用于运行时查询网络各层时间；

**RKNN_FLAG_MEM_ALLOC_OUTSIDE**：用于表示模型输入、输出、权重、中间tensor内存全部由用户分配，它主要有两方面的作用：

1. 所有内存均是用户自行分配，便于对整个系统内存进行统筹安排。
2. 用于内存复用，特别是针对RV1103/RV1106/RV1103B/RV1106B/RK2118这种内存极为紧张的情况。

假设有模型A、B 两个模型，这两个模型在设计上串行运行的，那么这两个模型的中间tensor的内存就可以复用。示例代码如下：

```
rknn_context ctx_a, ctx_b;

rknn_init(&ctx_a, model_path_a, 0, RKNN_FLAG_MEM_ALLOC_OUTSIDE, NULL);
rknn_query(ctx_a, RKNN_QUERY_MEM_SIZE, &mem_size_a, sizeof(mem_size_a));

rknn_init(&ctx_b, model_path_b, 0, RKNN_FLAG_MEM_ALLOC_OUTSIDE, NULL);
rknn_query(ctx_b, RKNN_QUERY_MEM_SIZE, &mem_size_b, sizeof(mem_size_b));
```

```
max_internal_size = MAX(mem_size_a.total_internal_size, mem_size_b.total_internal_size);
internal_mem_max = rknn_create_mem(ctx_a, max_internal_size);

internal_mem_a = rknn_create_mem_from_fd(ctx_a, internal_mem_max->fd,
        internal_mem_max->virt_addr, mem_size_a.total_internal_size, 0);
rknn_set_internal_mem(ctx_a, internal_mem_a);

internal_mem_b = rknn_create_mem_from_fd(ctx_b, internal_mem_max->fd,
        internal_mem_max->virt_addr, mem_size_b.total_internal_size, 0);
rknn_set_internal_mem(ctx_b, internal_mem_b);
```

**RKNN_FLAG_SHARE_WEIGHT_MEM**：用于共享另一个模型的weight权重。主要用于模拟不定长度模型输入（RKNPU运行时库版本大于等于1.5.0后该功能被动态shape功能替代）。比如对于某些语音模型，输入长度不定，但由于NPU无法支持不定长输入，因此需要生成几个不同分辨率的RKNN模，其中，只有一个RKNN模型的保留完整权重，其他RKNN模型不带权重。在初始化不带权重RKNN模型时，使用该标志能让当前上下文共享完整RKNN模型的权重。假设需要分辨率A、B两个模型，则使用流程如下：

1. 使用RKNN-Toolkit2生成分辨率A的模型。
2. 使用RKNN-Toolkit2生成不带权重的分辨率B的模型，rknn.config()中，remove_weight要设置成True，主要目的是减少模型B的大小。
3. 在板子上，正常初始化模型A。
4. 通过RKNN_FLAG_SHARE_WEIGHT_MEM的flags初始化模型B。
5. 其他按照原来的方式使用。板端参考代码如下：

```
rknn_context ctx_a, ctx_b;
rknn_init(&ctx_a, model_path_a, 0, 0, NULL);

rknn_init_extend extend;
extend.ctx = ctx_a;
rknn_init(&ctx_b, model_path_b, 0, RKNN_FLAG_SHARE_WEIGHT_MEM, &extend);
```

**RKNN_FLAG_COLLECT_MODEL_INFO_ONLY**：用于初始化一个空上下文，仅用于调用rknn_query接口查询模型weight内存总大小和中间tensor总大小，无法进行推理；

**RKNN_FLAG_INTERNAL_ALLOC_OUTSIDE**: 表示模型中间tensor由用户分配，常用于用户自行管理和复用多个模型之间的中间tensor内存；

**RKNN_FLAG_EXECUTE_FALLBACK_PRIOR_DEVICE_GPU**: 表示所有NPU不支持的层优先选择运行在GPU上，但并不保证运行在GPU上，实际运行的后端设备取决于运行时对该算子的支持情况；

**RKNN_FLAG_ENABLE_SRAM**: 表示中间tensor内存尽可能分配在SRAM上；

**RKNN_FLAG_SHARE_SRAM**: 用于当前上下文尝试共享另一个上下文的SRAM内存地址空间，要求当前上下文初始化时必须同时启用RKNN_FLAG_ENABLE_SRAM标志；

**RKNN_FLAG_DISABLE_PROC_HIGH_PRIORITY**: 表示当前上下文使用默认进程优先级。不设置该标志，进程nice值是-19；

**RKNN_FLAG_DISABLE_FLUSH_INPUT_MEM_CACHE**: 设置该标志后，runtime内部不主动刷新输入tensor缓存，用户必须确保输入tensor在调用 rknn_run 之前已刷新缓存。主要用于当输入数据没有CPU访问时，减少runtime内部刷cache的耗时。

**RKNN_FLAG_DISABLE_FLUSH_OUTPUT_MEM_CACHE**: 设置该标志后，runtime不主动清除输出tensor缓存。此时用户不能直接访问output_mem->virt_addr，这会导致缓存一致性问题。 如果用户想使用output_mem->virt_addr，必须使用 rknn_mem_sync (ctx, mem, RKNN_MEMORY_SYNC_FROM_DEVICE) 来刷新缓存。该标志一般在NPU的输出数据不被CPU访问时使用，比如输出数据由 GPU 或 RGA 访问以减少刷新缓存所需的时间。

**RKNN_FLAG_MODEL_BUFFER_ZERO_COPY**：表示rknn_init接口的传入的模型buffer是rknn_create_mem或者rknn_create_mem2接口分配的内存，runtime内部不需要拷贝一次模型buffer，减少运行时的内存占用，但需要用户保证上下文销毁前模型内存有效，并且在销毁上下文后释放该内存。初始化上下文时，rknn_init接口的rknn_init_extend参数，其成员real_model_offset、real_model_size、model_buffer_fd和model_buffer_flags根据rknn_create_mem2接口返回的rknn_tensor_mem设置。

**RKNN_MEM_FLAG_ALLOC_NO_CONTEXT**：在使用rknn_create_mem2接口分配内存时，设置该标志后，允许ctx参数是0或者NULL。从而用户在未初始化任何一个上下文之前就能获取NPU驱动分配的内存，返回的内存结构体需使用rknn_destroy_mem接口释放，释放的接口可使用任意一个上下文作为参数。示例代码如下：

```
  rknn_tensor_mem* model_mem = rknn_create_mem2(ctx, model_size,
RKNN_MEM_FLAG_ALLOC_NO_CONTEXT);
  memcpy(model_mem->virt_addr, model_data, model_size);
  rknn_init_extend init_ext;
  memset(&init_ext, 0, sizeof(rknn_init_extend));
  init_ext.real_model_offset  = 0;
  init_ext.real_model_size    = model_size;
  init_ext.model_buffer_fd    = model_mem->fd;
  init_ext.model_buffer_flags = model_mem->flags;
  int ret = rknn_init(&ctx, model_mem->virt_addr, model_size,
RKNN_FLAG_MODEL_BUFFER_ZERO_COPY, &init_ext);

  // do rknn inference...

  rknn_destroy_mem(ctx, model_mem);
  rknn_destroy(ctx);
```

## rknn_set_core_mask

rknn_set_core_mask函数指定工作的NPU核心，该函数仅支持RK3576/RK3588平台，在单核NPU架构的平台上设置会返回错误。

| API | rknn_set_core_mask |
| --- | --- |
| 功能 | 设置运行的NPU核心。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | rknn_core_mask core_mask：NPU核心的枚举类型，目前有如下方式配置：<br>**RKNN_NPU_CORE_AUTO**：表示自动调度模型，自动运行在当前空闲的NPU核上；<br>**RKNN_NPU_CORE_0**：表示运行在NPU0核上；<br>**RKNN_NPU_CORE_1**：表示运行在NPU1核上；<br>**RKNN_NPU_CORE_2**：表示运行在NPU2核上；<br>**RKNN_NPU_CORE_0_1**：表示同时工作在NPU0、NPU1核上；<br>**RKNN_NPU_CORE_0_1_2**：表示同时工作在NPU0、NPU1、NPU2核上；<br>**RKNN_NPU_CORE_ALL**：表示工作在所有的NPU核心上 |

| API | rknn_set_core_mask |
| --- | --- |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_context ctx;
rknn_core_mask core_mask = RKNN_NPU_CORE_0;
int ret = rknn_set_core_mask(ctx, core_mask);
```

在RKNN_NPU_CORE_0_1及RKNN_NPU_CORE_0_1_2模式下，目前以下OP能获得更好的加速：Conv、DepthwiseConvolution、Add、Concat、Relu、Clip、Relu6、ThresholdedRelu、PRelu、LeakyRelu，其余类型OP将fallback至单核Core0中运行，部分类型OP（如Pool类、ConvTranspose等）将在后续更新版本中支持。

## rknn_set_batch_core_num

rknn_set_batch_core_num函数指定多batch RKNN模型（RKNN-Toolkit2转换时设置rknn_batch_size大于1导出的模型）的NPU核心数量，该函数仅支持RK3588/RK3576平台。

| API | rknn_set_batch_core_num |
| --- | --- |
| 功能 | 设置多batch RKNN模型运行的NPU核心数量。 |
| 参数 | rknn_context context：rknn_context对象。 |
|  | int core_num：指定运行的核心数量。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_context ctx;
int ret = rknn_set_batch_core_num(ctx, 2);
```

## rknn_dup_context

rknn_dup_context生成一个指向同一个模型的新context，可用于多线程执行相同模型时的权重复用。
**RV1106/RV1103/RV1106B/RV1103B/RK2118平台暂不支持**。

| API | rknn_dup_context |
| --- | --- |
| 功能 | 生成同一个模型的两个ctx，复用模型的权重信息。 |
| 参数 | rknn_context * context_in：rknn_context指针。初始化后的rknn_context对象。 |
|  | rknn_context * context_out：rknn_context指针。生成新的rknn_context对象。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_context ctx_in;
rknn_context ctx_out;
int ret = rknn_dup_context(&ctx_in, &ctx_out);
```

## rknn_destroy

rknn_destroy函数将释放传入的rknn_context及其相关资源。

| API | rknn_destroy |
| --- | --- |
| 功能 | 销毁rknn_context对象及其相关资源。 |
| 参数 | rknn_context context：要销毁的rknn_context对象。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_context ctx;
int ret = rknn_destroy(ctx);
```

## rknn_query

rknn_query函数能够查询获取到模型输入输出信息、逐层运行时间、模型推理的总时间、SDK版本、内存占用信息、用户自定义字符串等信息。

| API | rknn_query |
| --- | --- |
| 功能 | 查询模型与SDK的相关信息。 |
| 参数 | rknn_context context：rknn_context对象。 |
|  | rknn_query_cmd ： 查询命令。 |
|  | void* info：存放返回结果的结构体变量。 |
|  | uint32_t size：info对应的结构体变量的大小。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

当前SDK支持的查询命令如下表所示：

| 查询命令 | 返回结果结构体 | 功能 |
| --- | --- | --- |
| RKNN_QUERY_IN_OUT_NUM | rknn_input_output_num | 查询输入输出tensor个数。 |
| RKNN_QUERY_INPUT_ATTR | rknn_tensor_attr | 查询输入tensor属性。 |
| RKNN_QUERY_OUTPUT_ATTR | rknn_tensor_attr | 查询输出tensor属性。 |
| RKNN_QUERY_PERF_DETAIL | rknn_perf_detail | 查询网络各层运行时间，需要调用rknn_init接口时，设置RKNN_FLAG_COLLECT_PERF_MASK标志才能生效。 |
| RKNN_QUERY_PERF_RUN | rknn_perf_run | 查询推理模型（不包含设置输入/输出）的耗时，单位是微秒。 |
| RKNN_QUERY_SDK_VERSION | rknn_sdk_version | 查询SDK版本。 |

| 查询命令 | 返回结果结构体 | 功能 |
| --- | --- | --- |
| RKNN_QUERY_MEM_SIZE | rknn_mem_size | 查询分配给权重和网络中间tensor的内存大小。 |
| RKNN_QUERY_CUSTOM_STRING | rknn_custom_string | 查询RKNN模型里面的用户自定义字符串信息。 |
| RKNN_QUERY_NATIVE_INPUT_ATTR | rknn_tensor_attr | 使用零拷贝API接口时,查询原生输入tensor属性，它是NPU直接读取的模型输入属性。 |
| RKNN_QUERY_NATIVE_OUTPUT_ATTR | rknn_tensor_attr | 使用零拷贝API接口时,查询原生输出tensor属性，它是NPU直接输出的模型输出属性。 |
| RKNN_QUERY_NATIVE_NC1HWC2_INPUT_ATTR | rknn_tensor_attr | 使用零拷贝API接口时,查询原生输入tensor属性，它是NPU直接读取的模型输入属性与RKNN_QUERY_NATIVE_INPUT_ATTR查询结果一致。 |
| RKNN_QUERY_NATIVE_NC1HWC2_OUTPUT_ATTR | rknn_tensor_attr | 使用零拷贝API接口时,查询原生输出tensor属性，它是NPU直接输出的模型输出属与RKNN_QUERY_NATIVE_OUTPUT_ATTR查询结果一致性。 |
| RKNN_QUERY_NATIVE_NHWC_INPUT_ATTR | rknn_tensor_attr | 使用零拷贝API接口时,查询原生输入 tensor属性与RKNN_QUERY_NATIVE_INPUT_ATTR查询结果一致 。 |
| RKNN_QUERY_NATIVE_NHWC_OUTPUT_ATTR | rknn_tensor_attr | 使用零拷贝API接口时,查询原生输出NHWC tensor属性。 |
| RKNN_QUERY_DEVICE_MEM_INFO | rknn_tensor_mem | 查询模型buffer的内存属性。 |
| RKNN_QUERY_INPUT_DYNAMIC_RANGE | rknn_input_range | 使用支持动态形状RKNN模型时，查询模型支持输入形状数量、列表、形状对应的数据布局和名称等信息。 |
| RKNN_QUERY_CURRENT_INPUT_ATTR | rknn_tensor_attr | 使用支持动态形状RKNN模型时，查询模型当前推理所使用的输入属性。 |
| RKNN_QUERY_CURRENT_OUTPUT_ATTR | rknn_tensor_attr | 使用支持动态形状RKNN模型时，查询模型当前推理所使用的输出属性。 |
| RKNN_QUERY_CURRENT_NATIVE_INPUT_ATTR | rknn_tensor_attr | 使用支持动态形状RKNN模型时，查询模型当前推理所使用的NPU原生输入属性。 |
| RKNN_QUERY_CURRENT_NATIVE_OUTPUT_ATTR | rknn_tensor_attr | 使用支持动态形状RKNN模型时，查询模型当前推理所使用的NPU原生输出属性。 |

各个指令用法的详细说明，如下：

1. **查询SDK版本**

   传入RKNN_QUERY_SDK_VERSION命令可以查询RKNN SDK的版本信息。其中需要先创建rknn_sdk_version结构体对象。

   示例代码如下：

```
rknn_sdk_version version;
ret = rknn_query(ctx, RKNN_QUERY_SDK_VERSION, &version, sizeof(rknn_sdk_version));
printf("sdk api version: %s\n", version.api_version);
printf("driver version: %s\n", version.drv_version);
```

2. **查询输入输出tensor个数**

   在rknn_init接口调用完毕后，传入RKNN_QUERY_IN_OUT_NUM命令可以查询模型输入输出tensor的个数。其中需要先创建rknn_input_output_num结构体对象。

   示例代码如下：

```
rknn_input_output_num io_num;
ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &io_num, sizeof(io_num));
printf("model input num: %d, output num: %d\n", io_num.n_input, io_num.n_output);
```

3. **查询输入tensor属性(用于通用API接口)**

在rknn_init接口调用完毕后，传入RKNN_QUERY_INPUT_ATTR命令可以查询模型输入tensor的属性。其中需要先创建rknn_tensor_attr结构体对象 (**注意：RV1106/RV1103/RV1106B/RV1103B/RK2118查询出来的tensor是原始输入native的tensor**) 。

示例代码如下：

```
rknn_tensor_attr input_attrs[io_num.n_input];
memset(input_attrs, 0, sizeof(input_attrs));
for (int i = 0; i < io_num.n_input; i++) {
    input_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_INPUT_ATTR, &(input_attrs[i]),
sizeof(rknn_tensor_attr));
}
```

4. **查询输出tensor属性(用于通用API接口)**

在rknn_init接口调用完毕后，传入RKNN_QUERY_OUTPUT_ATTR命令可以查询模型输出tensor的属性。其中需要先创建rknn_tensor_attr结构体对象。

示例代码如下：

```
rknn_tensor_attr output_attrs[io_num.n_output];
memset(output_attrs, 0, sizeof(output_attrs));
for (int i = 0; i < io_num.n_output; i++) {
    output_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &(output_attrs[i]),
sizeof(rknn_tensor_attr));
}
```

5. **查询模型推理的逐层耗时**

在rknn_run接口调用完毕后，rknn_query接口传入RKNN_QUERY_PERF_DETAIL可以查询网络推理时逐层的耗时，单位是微秒。使用该命令的前提是，在rknn_init接口的flag参数需要包含RKNN_FLAG_COLLECT_PERF_MASK标志。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, RKNN_FLAG_COLLECT_PERF_MASK,
NULL);
...
ret = rknn_run(ctx,NULL);
...
rknn_perf_detail perf_detail;
ret = rknn_query(ctx, RKNN_QUERY_PERF_DETAIL, &perf_detail, sizeof(perf_detail));
```

6. **查询模型推理的总耗时**

在rknn_run接口调用完毕后，rknn_query接口传入RKNN_QUERY_PERF_RUN可以查询上模型推理（不包含设置输入/输出）的耗时，单位是微秒。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0, NULL);
...
ret = rknn_run(ctx,NULL);
...
rknn_perf_run perf_run;
ret = rknn_query(ctx, RKNN_QUERY_PERF_RUN, &perf_run, sizeof(perf_run));
```

7. **查询模型的内存占用情况**

   在rknn_init接口调用完毕后，当用户需要自行分配网络的内存时，rknn_query接口传入RKNN_QUERY_MEM_SIZE可以查询模型的权重、网络中间tensor的内存（不包括输入和输出）、推演模型所用的所有DMA内存的以及SRAM内存（如果sram没开或者没有此项功能则为0）的占用情况。使用该命令的前提是在rknn_init接口的flag参数需要包含RKNN_FLAG_MEM_ALLOC_OUTSIDE标志。

   示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, RKNN_FLAG_MEM_ALLOC_OUTSIDE ,
NULL);
rknn_mem_size mem_size;
ret = rknn_query(ctx, RKNN_QUERY_MEM_SIZE, &mem_size, sizeof(mem_size));
```

8. **查询模型中用户自定义字符串**

   在rknn_init接口调用完毕后，当用户需要查询生成RKNN模型时加入的自定义字符串，rknn_query接口传入RKNN_QUERY_CUSTOM_STRING可以获取该字符串。例如，在转换RKNN模型时，用户填入“RGB”的自定义字符来标识RKNN模型输入是RGB格式三通道图像而不是BGR格式三通道图像，在运行时则根据查询到的“RGB”信息将数据转换成RGB图像。

   示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0, NULL);
rknn_custom_string custom_string;
ret = rknn_query(ctx, RKNN_QUERY_CUSTOM_STRING, &custom_string, sizeof(custom_string));
```

9. **查询原生输入tensor属性(用于零拷贝API接口)**

   在rknn_init接口调用完毕后，传入RKNN_QUERY_NATIVE_INPUT_ATTR命令（同RKNN_QUERY_NATIVE_NC1HWC2_INPUT_ATTR）可以查询模型原生输入tensor的属性。其中需要先创建rknn_tensor_attr结构体对象。

   示例代码如下：

```
rknn_tensor_attr input_attrs[io_num.n_input];
memset(input_attrs, 0, sizeof(input_attrs));
for (int i = 0; i < io_num.n_input; i++) {
    input_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_NATIVE_INPUT_ATTR, &(input_attrs[i]),
sizeof(rknn_tensor_attr));
}
```

10. **查询原生输出tensor属性(用于零拷贝API接口)**

在rknn_init接口调用完毕后，传入RKNN_QUERY_NATIVE_OUTPUT_ATTR命令（同RKNN_QUERY_NATIVE_NC1HWC2_OUTPUT_ATTR）可以查询模型原生输出tensor的属性。其中需要先创建rknn_tensor_attr结构体对象。

示例代码如下：

```
rknn_tensor_attr output_attrs[io_num.n_output];
memset(output_attrs, 0, sizeof(output_attrs));
for (int i = 0; i < io_num.n_output; i++) {
    output_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_NATIVE_OUTPUT_ATTR, &(output_attrs[i]),
sizeof(rknn_tensor_attr));
}
```

11. **查询NHWC格式原生输入tensor属性(用于零拷贝API接口)**

    在rknn_init接口调用完毕后，传入RKNN_QUERY_NATIVE_NHWC_INPUT_ATTR命令可以查询模型**NHWC格式**输入tensor的属性。其中需要先创建rknn_tensor_attr结构体对象。

    示例代码如下：

```
rknn_tensor_attr input_attrs[io_num.n_input];
memset(input_attrs, 0, sizeof(input_attrs));
for (int i = 0; i < io_num.n_input; i++) {
    input_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_NATIVE_NHWC_INPUT_ATTR,
                     &(input_attrs[i]), sizeof(rknn_tensor_attr));
}
```

12. **查询NHWC格式原生输出tensor属性(用于零拷贝API接口)**

    在rknn_init接口调用完毕后，传入RKNN_QUERY_NATIVE_NHWC_OUTPUT_ATTR命令可以查询模型**NHWC格式**输出tensor的属性。其中需要先创建rknn_tensor_attr结构体对象。

    示例代码如下：

```
rknn_tensor_attr output_attrs[io_num.n_output];
memset(output_attrs, 0, sizeof(output_attrs));
for (int i = 0; i < io_num.n_output; i++) {
    output_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_NATIVE_NHWC_OUTPUT_ATTR,
                     &(output_attrs[i]), sizeof(rknn_tensor_attr));
}
```

13. **查询模型buffer的内存属性（注：仅RV1106/RV1103/RV1106B/RV1103B/RK2118支持该查询）**
    在rknn_init接口调用完毕后，传入RKNN_QUERY_DEVICE_MEM_INFO命令可以查询Runtime内部开辟的模型buffer的包括fd、物理地址等属性。

```
rknn_tensor_mem mem_info;
memset(&mem_info, 0, sizeof(mem_info));
ret = rknn_query(ctx, RKNN_QUERY_DEVICE_MEM_INFO, &mem_info, sizeof(mem_info));
```

14. **查询RKNN模型支持的动态输入形状信息（注：RV1106/RV1103/RV1106B/RV1103B/RK2118不支持该接口）**

在rknn_init接口调用完毕后，传入RKNN_QUERY_INPUT_DYNAMIC_RANGE命令可以查询模型支持的输入形状信息，包含输入形状个数 、输入形状列表、输入形状对应的布局和名称等信息。其中需要先创建rknn_input_range结构体对象。

示例代码如下：

```
rknn_input_range dyn_range[io_num.n_input];
memset(dyn_range, 0, io_num.n_input * sizeof(rknn_input_range));
for (uint32_t i = 0; i < io_num.n_input; i++) {
    dyn_range[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_INPUT_DYNAMIC_RANGE,
                     &dyn_range[i], sizeof(rknn_input_range));
}
```

15. **查询RKNN模型当前使用的输入动态形状**

在rknn_set_input_shapes接口调用完毕后，传入RKNN_QUERY_CURRENT_INPUT_ATTR命令可以查询模型当前使用的输入属性信息。其中需要先创建rknn_tensor_attr结构体（注：RV1106/RV1103/RV1106B/RV1103B/RK2118不支持该命令）。

示例代码如下：

```
rknn_tensor_attr cur_input_attrs[io_num.n_input];
memset(cur_input_attrs, 0, io_num.n_input * sizeof(rknn_tensor_attr));
for (uint32_t i = 0; i < io_num.n_input; i++) {
    cur_input_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_CURRENT_INPUT_ATTR,
                     &(cur_input_attrs[i]), sizeof(rknn_tensor_attr));
}
```

16. **查询RKNN模型当前使用的输出动态形状**

在rknn_set_input_shapes接口调用完毕后，传入RKNN_QUERY_CURRENT_OUTPUT_ATTR命令可以查询模型当前使用的输出属性信息。其中需要先创建rknn_tensor_attr结构体（注：RV1106/RV1103/RV1106B/RV1103B/RK2118不支持该命令）。

示例代码如下：

```
rknn_tensor_attr cur_output_attrs[io_num.n_output];
memset(cur_output_attrs, 0, io_num.n_output * sizeof(rknn_tensor_attr));
for (uint32_t i = 0; i < io_num.n_output; i++) {
    cur_output_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_CURRENT_OUTPUT_ATTR,
                     &(cur_output_attrs[i]), sizeof(rknn_tensor_attr));
}
```

17. **查询RKNN模型当前使用的原生输入动态形状**

在rknn_set_input_shapes接口调用完毕后，传入RKNN_QUERY_CURRENT_NATIVE_INPUT_ATTR命令可以查询模型当前使用的原生输入属性信息。其中需要先创建rknn_tensor_attr结构体（注：RV1106/RV1103/RV1106B/RV1103B/RK2118不支持该命令）。

示例代码如下：

```
rknn_tensor_attr cur_input_attrs[io_num.n_input];
memset(cur_input_attrs, 0, io_num.n_input * sizeof(rknn_tensor_attr));
for (uint32_t i = 0; i < io_num.n_input; i++) {
    cur_input_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_CURRENT_NATIVE_INPUT_ATTR,
                     &(cur_input_attrs[i]), sizeof(rknn_tensor_attr));
}
```

18. **查询RKNN模型当前使用的原生输出动态形状**

    在rknn_set_input_shapes接口调用完毕后，传入RKNN_QUERY_CURRENT_NATIVE_OUTPUT_ATTR命令可以查询模型当前使用的原生输出属性信息。其中需要先创建rknn_tensor_attr结构体（注：RV1106/RV1103/RV1106B/RV1103B/RK2118不支持该命令）。

    示例代码如下：

```
rknn_tensor_attr cur_output_attrs[io_num.n_output];
memset(cur_output_attrs, 0, io_num.n_output * sizeof(rknn_tensor_attr));
for (uint32_t i = 0; i < io_num.n_output; i++) {
    cur_output_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_CURRENT_NATIVE_OUTPUT_ATTR,
                     &(cur_output_attrs[i]), sizeof(rknn_tensor_attr));
}
```

## rknn_inputs_set

通过rknn_inputs_set函数可以设置模型的输入数据。该函数能够支持多个输入，其中每个输入是rknn_input结构体对象，在传入之前用户需要设置该对象，**注：RV1106/RV1103/RV1106B/RV1103B/RK2118不支持该接口**。

| API | rknn_inputs_set |
| --- | --- |
| 功能 | 设置模型输入数据。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | uint32_t n_inputs：输入数据个数。 |
| | rknn_input inputs[]：输入数据数组，数组每个元素是rknn_input结构体对象。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_input inputs[1];
memset(inputs, 0, sizeof(inputs));
inputs[0].index = 0;
inputs[0].type = RKNN_TENSOR_UINT8;
inputs[0].size = img_width*img_height*img_channels;
inputs[0].fmt = RKNN_TENSOR_NHWC;
inputs[0].buf = in_data;
inputs[0].pass_through = 0;

ret = rknn_inputs_set(ctx, 1, inputs);
```

## rknn_run

rknn_run函数将执行一次模型推理，调用之前需要先通过rknn_inputs_set函数或者零拷贝的接口设置输入数据。

| API | rknn_run |
| --- | --- |
| 功能 | 执行一次模型推理。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | rknn_run_extend* extend：保留扩展，当前没有使用，传入NULL即可。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
ret = rknn_run(ctx, NULL);
```

## rknn_outputs_get

rknn_outputs_get函数可以获取模型推理的输出数据。该函数能够一次获取多个输出数据。其中每个输出是rknn_output结构体对象，在函数调用之前需要依次创建并设置每个rknn_output对象。

对于输出数据的buffer存放可以采用两种方式：一种是用户自行申请和释放，此时rknn_output对象的is_prealloc需要设置为1，并且将buf指针指向用户申请的buffer；另一种是由rknn来进行分配，此时rknn_output对象的is_prealloc设置为0即可，函数执行之后buf将指向输出数据。**注：RV1106/RV1103/RV1106B/RV1103B/RK2118不支持该接口**

| API | rknn_outputs_get |
| --- | --- |
| 功能 | 获取模型推理输出。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | uint32_t n_outputs：输出数据个数。 |
| | rknn_output outputs[]：输出数据的数组，其中数组每个元素为rknn_output结构体对象，代表模型的一个输出。 |
| | rknn_output_extend* extend：保留扩展，当前没有使用，传入NULL即可。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_output outputs[io_num.n_output];
memset(outputs, 0, sizeof(outputs));
for (int i = 0; i < io_num.n_output; i++) {
    outputs[i].index = i;
    outputs[i].is_prealloc = 0;
    outputs[i].want_float = 1;
}
ret = rknn_outputs_get(ctx, io_num.n_output, outputs, NULL);
```

## rknn_outputs_release

rknn_outputs_release函数将释放rknn_outputs_get函数得到的输出的相关资源。

| API | rknn_outputs_release |
| --- | --- |
| 功能 | 释放rknn_output对象。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | uint32_t n_outputs：输出数据个数。 |
| | rknn_output outputs[]：要销毁的rknn_output数组。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
ret = rknn_outputs_release(ctx, io_num.n_output, outputs);
```

## rknn_create_mem_from_phys

当用户需要自己分配内存让NPU使用时，通过rknn_create_mem_from_phys函数可以创建一个rknn_tensor_mem结构体并得到它的指针，该函数通过传入物理地址、虚拟地址以及大小，外部内存相关的信息会赋值给rknn_tensor_mem结构体。

| API | rknn_create_mem_from_phys |
| --- | --- |
| 功能 | 通过物理地址创建rknn_tensor_mem结构体并分配内存。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | uint64_t phys_addr：内存的物理地址。 |
| | void *virt_addr：内存的虚拟地址。 |
| | uint32_t size：内存的大小。 |
| 返回值 | rknn_tensor_mem*： tensor内存信息结构体指针。 |

示例代码如下：

```
//suppose we have got buffer information as input_phys, input_virt and size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem_from_phys(ctx, input_phys, input_virt, size);
```

## rknn_create_mem_from_fd

当用户要自己分配内存让NPU使用时，rknn_create_mem_from_fd函数可以创建一个rknn_tensor_mem结构体并得到它的指针，该函数通过传入文件描述符fd、偏移、虚拟地址以及大小，外部内存相关的信息会赋值给rknn_tensor_mem结构体。

| API | rknn_create_mem_from_fd |
| --- | --- |
| 功能 | 通过文件描述符创建rknn_tensor_mem结构体。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | int32_t fd：内存的文件描述符。 |
| | void *virt_addr：内存的虚拟地址，**fd对应的内存的首地址**。 |
| | uint32_t size：内存的大小。 |
| | int32_t offset：内存相对于文件描述符和虚拟地址的偏移量。 |
| 返回值 | rknn_tensor_mem*： tensor内存信息结构体指针。 |

示例代码如下：

```
//suppose we have got buffer information as input_fd, input_virt and size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem_from_fd(ctx, input_fd, input_virt, size, 0);
```

## rknn_create_mem

当用户要NPU内部分配内存时，rknn_create_mem函数可以分配用户指定的内存大小，并返回一个rknn_tensor_mem结构体。

| API | rknn_create_mem |
| --- | --- |
| 功能 | 创建rknn_tensor_mem结构体并分配内存。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | uint32_t size：内存的大小。 |
| 返回值 | rknn_tensor_mem*： tensor内存信息结构体指针。 |

示例代码如下：

```
//suppose we have got buffer size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem(ctx, size);
```

## rknn_create_mem2

当用户要NPU内部分配内存时，rknn_create_mem2函数可以分配用户指定的内存大小及内存类型，并返回一个rknn_tensor_mem结构体。

| API | rknn_create_mem2 |
| --- | --- |
| 功能 | 创建rknn_tensor_mem结构体并分配内存。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | uint64_t size：内存的大小。 |
| | uint64_t alloc_flags: 控制内存是否是cacheable的。 RKNN_FLAG_MEMORY_CACHEABLE：创建cacheable内存 RKNN_FLAG_MEMORY_NON_CACHEABLE: 创建non -cacheable内存 RKNN_FLAG_MEMORY_FLAGS_DEFAULT：同RKNN_FLAG_MEMORY_CACHEABLE |
| 返回值 | rknn_tensor_mem*： tensor内存信息结构体指针。 |

rknn_create_mem2与rknn_create_mem的主要区别是rknn_create_mem2带了一个alloc_flags，可以指定分配的内存是否cacheable的，而rknn_create_mem不能指定，默认就是cacheable。

示例代码如下：

```
//suppose we have got buffer size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem2(ctx, size, RKNN_FLAG_MEMORY_NON_CACHEABLE);
```

## rknn_destroy_mem

rknn_destroy_mem函数会销毁rknn_tensor_mem结构体，用户分配的内存需要自行释放。

| API | rknn_destroy_mem |
| --- | --- |
| 功能 | 销毁rknn_tensor_mem结构体。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | rknn_tensor_mem*：tensor内存信息结构体指针。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_tensor_mem* input_mems [1];
int ret = rknn_destroy_mem(ctx, input_mems[0]);
```

## rknn_set_weight_mem

如果用户自己为网络权重分配内存，初始化相应的rknn_tensor_mem结构体后，在调用rknn_run前，通过rknn_set_weight_mem函数可以让NPU使用该内存。

| API | rknn_set_weight_mem |
| --- | --- |
| 功能 | 设置包含权重内存信息的rknn_tensor_mem结构体。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | rknn_tensor_mem*：权重tensor内存信息结构体指针。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_tensor_mem* weight_mems [1];
int ret = rknn_set_weight_mem(ctx, weight_mems[0]);
```

## rknn_set_internal_mem

如果用户自己为网络中间tensor分配内存，初始化相应的rknn_tensor_mem结构体后，在调用rknn_run前，通过rknn_set_internal_mem函数可以让NPU使用该内存。

| API | rknn_set_internal_mem |
| --- | --- |
| 功能 | 设置包含中间tensor内存信息的rknn_tensor_mem结构体。 |

| API | rknn_set_internal_mem |
| --- | --- |
| 参数 | rknn_context context：rknn_context对象。 |
| | rknn_tensor_mem*：模型中间tensor内存信息结构体指针。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_tensor_mem* internal_tensor_mems [1];
int ret = rknn_set_internal_mem(ctx, internal_tensor_mems[0]);
```

## rknn_set_io_mem

如果用户自己为网络输入/输出tensor分配内存，初始化相应的rknn_tensor_mem结构体后，在调用rknn_run前，通过rknn_set_io_mem函数可以让NPU使用该内存。

| API | rknn_set_io_mem |
| --- | --- |
| 功能 | 设置包含模型输入/输出内存信息的rknn_tensor_mem结构体。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | rknn_tensor_mem*：输入/输出tensor内存信息结构体指针。 |
| | rknn_tensor_attr *: 输入/输出tensor的属性。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_tensor_attr output_attrs[1];
rknn_tensor_mem* output_mems[1];
ret = rknn_query(ctx, RKNN_QUERY_NATIVE_OUTPUT_ATTR, &(output_attrs[0]),
sizeof(rknn_tensor_attr));
output_mems[0] = rknn_create_mem(ctx, output_attrs[0].size_with_stride);
rknn_set_io_mem(ctx, output_mems[0], &output_attrs[0]);
```

## rknn_set_input_shape（deprecated）

该接口已经废弃，请使用rknn_set_input_shapes接口绑定输入形状。当前版本不可用，如要继续使用该接口，请使用1.5.0版本SDK并参考1.5.0版本的使用指南文档。

## rknn_set_input_shapes

对于动态形状输入RKNN模型，在推理前必须指定当前使用的输入形状。该接口传入输入个数和rknn_tensor_attr数组，包含了每个输入形状和对应的数据布局信息，将每个rknn_tensor_attr结构体对象的索引、名称、形状（dims）和内存布局信息（fmt）必须填充，rknn_tensor_attr结构体其他成员无需设置。在使用该接口前，可先通过rknn_query函数查询RKNN模型支持的输入形状数量和动态形状列表，要求输入数据的形状在模型支持的输入形状列表中。初次运行或每次切换新的输入形状，需要调用该接口设置新的形状，否则，不需要重复调用该接口。

| API | rknn_set_input_shapes |
| --- | --- |
| 功能 | 设置模型当前使用的输入形状。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | uint32_t n_inputs：输入Tensor的数量。 |
| | rknn_tensor_attr *: 输入tensor的属性数组指针，传递所有输入的形状信息，用户需要设置每个输入属性结构体中的index、name、dims、fmt、n_dims成员，其他成员无需设置。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
for (int i = 0; i < io_num.n_input; i++) {
    for (int j = 0; j < input_attrs[i].n_dims; ++j) {
//使用第一个动态输入形状
        input_attrs[i].dims[j] = dyn_range[i].dyn_range[0][j];
    }
}
ret = rknn_set_input_shapes(ctx, io_num.n_input, input_attrs);
if (ret < 0) {
    fprintf(stderr, "rknn_set_input_shapes error! ret=%d\n", ret);
    return -1;
}
```

## rknn_mem_sync

rknn_create_mem函数创建的内存默认是带cacheable标志的，对于带cacheable标志创建的内存，在被CPU和NPU同时使用时，由于cache行为会导致数据一致性问题。该接口用于同步一块带cacheable标志创建的内存，保证CPU和NPU访问这块内存的数据是一致的。

| API | rknn_mem_sync |
| --- | --- |
| 功能 | 同步CPU cache和DDR数据。 |
| 参数 | rknn_context context：rknn_context对象。 |
| | rknn_tensor_mem* mem：tensor内存信息结构体指针。 |
| | rknn_mem_sync_mode mode: 表示刷新CPU cache和DDR数据的模式。<br>**RKNN_MEMORY_SYNC_TO_DEVICE**：表示CPU cache数据同步到DDR中，通常用于CPU写入内存后，NPU访问相同内存前使用该模式将cache中的数据写回DDR。<br>**RKNN_MEMORY_SYNC_FROM_DEVICE**：表示DDR数据同步到CPU cache，通常用于NPU写入内存后，使用该模式让下次CPU访问相同内存时，cache数据无效，CPU从DDR重新读取数据。<br>**RKNN_MEMORY_SYNC_BIDIRECTIONAL**：表示CPU cache数据同步到DDR同时令CPU 重新从DDR读取数据。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
ret =rknn_mem_sync(ctx, &outputs[0].mem,
RKNN_MEMORY_SYNC_FROM_DEVICE);
if (ret < 0) {
    fprintf(stderr, " rknn_mem_sync error! ret=%d\n", ret);
    return -1;
}
```

## 4.4 矩阵乘法数据结构定义

### rknn_matmul_info

rknn_matmul_info表示用于执行矩阵乘法的规格信息，它包含了矩阵乘法的规模、输入和输出矩阵的数据类型和内存排布。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| M | int32_t | A矩阵的行数 |
| K | int32_t | A矩阵的列数 |
| N | int32_t | B矩阵的列数 |
| type | rknn_matmul_type | 输入输出矩阵的数据类型：<br>**RKNN_FLOAT16_MM_FLOAT16_TO_FLOAT32**：表示矩阵A和B是float16类型，矩阵C是float类型；<br>**RKNN_INT8_MM_INT8_TO_INT32**：表示矩阵A和B是int8类型，矩阵C是int32类型； **RKNN_INT8_MM_INT8_TO_INT8**：表示矩阵A、B和C是int8类型； **RKNN_FLOAT16_MM_FLOAT16_TO_FLOAT16**：表示矩阵A、B和C是float16类型；<br>**RKNN_FLOAT16_MM_INT8_TO_FLOAT32**：表示矩阵A是float16类型，矩阵B是int8类型，矩阵C是float类型；<br>**RKNN_FLOAT16_MM_INT8_TO_FLOAT16**：表示矩阵A是float16类型，矩阵B是int8类型，矩阵C是float16类型；<br>**RKNN_FLOAT16_MM_INT4_TO_FLOAT32**：表示矩阵A是float16类型，矩阵B是int4类型，矩阵C是float类型；<br>**RKNN_FLOAT16_MM_INT4_TO_FLOAT16**：表示矩阵A是float16类型，矩阵B是int4类型，矩阵C是float16类型；<br>**RKNN_INT8_MM_INT8_TO_FLOAT32**：表示矩阵A和B是int8类型，矩阵C是float类型;<br>**RKNN_INT4_MM_INT4_TO_INT16**：表示矩阵A和B是int4类型，矩阵C是int16类型； **RKNN_INT8_MM_INT4_TO_INT32**：表示矩阵A是int8类型，B是int4类型，矩阵C是int32类型 |
| B_layout | int16_t | 矩阵B的数据排列方式。<br>**RKNN_MM_LAYOUT_NORM**：表示矩阵B按照原始形状排列 ，即KxN的形状排列；<br>**RKNN_MM_LAYOUT_NATIVE**：表示矩阵B按照高性能形状排列；<br>**RKNN_MM_LAYOUT_TP_NORM**：表示矩阵B按照Transpose后的形状排列，即NxK的形状排列 |

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| B_quant_type | int16_t | 矩阵B的量化方式类型。<br>**RKNN_QUANT_TYPE_PER_LAYER_SYM**：表示矩阵B按照Per-Layer方式对称量化；<br>**RKNN_QUANT_TYPE_PER_LAYER_ASYM**：表示矩阵B按照Per-Layer方式非对称量化；<br>**RKNN_QUANT_TYPE_PER_CHANNEL_SYM**：表示矩阵B按照Per-Channel方式对称量化；<br>**RKNN_QUANT_TYPE_PER_CHANNEL_ASYM**：表示矩阵B按照Per-Channel方式非对称量化；<br>**RKNN_QUANT_TYPE_PER_GROUP_SYM**：表示矩阵B按照Per-Group方式对称量化；<br>**RKNN_QUANT_TYPE_PER_GROUP_ASYM**：表示矩阵B按照Per-Group方式非对称量化 |
| AC_layout | int16_t | 矩阵A和C的数据排列方式。<br>**RKNN_MM_LAYOUT_NORM**：表示矩阵A和C按照原始形状排列；<br>**RKNN_MM_LAYOUT_NATIVE**：表示矩阵A和C按照高性能形状排列 |
| AC_quant_type | int16_t | 矩阵A和C的量化方式类型。<br>**RKNN_QUANT_TYPE_PER_LAYER_SYM**：表示矩阵A和C按照Per-Layer方式对称量化；<br>**RKNN_QUANT_TYPE_PER_LAYER_ASYM**：表示矩阵A和C按照Per-Layer方式非对称量化 |
| iommu_domain_id | int32_t | 矩阵上下文所在的IOMMU地址空间域的索引。IOMMU地址空间与上下文一一对应，每个IOMMU地址空间大小为4GB。该参数主要用于矩阵A、B和C的参数规格较大，某个域内NPU分配的内存超过4GB以后需切换另一个域时使用。 |
| group_size | int16_t | 一个组的元素数量，仅分组量化开启后生效。 |
| reserved | int8_t[] | 预留字段 |

## rknn_matmul_tensor_attr

rknn_matmul_tensor_attr表示每个矩阵tensor的属性，它包含了矩阵的名字、形状、大小和数据类型。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| name | char[] | 矩阵的名字 |
| n_dims | uint32_t | 矩阵的维度个数 |
| dims | uint32_t[] | 矩阵的形状 |
| size | uint32_t | 矩阵的大小，以字节为单位 |
| type | rknn_tensor_type | 矩阵的数据类型 |

## rknn_matmul_io_attr

rknn_matmul_io_attr表示矩阵所有输入和输出tensor的属性，它包含了矩阵A、B和C的属性。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| A | rknn_matmul_tensor_attr | 矩阵A的tensor属性 |
| B | rknn_matmul_tensor_attr | 矩阵B的tensor属性 |
| C | rknn_matmul_tensor_attr | 矩阵C的tensor属性 |

## rknn_quant_params

rknn_quant_params表示矩阵的量化参数，包括name以及scale和zero_point数组的指针和长度，name用来标识矩阵的名称，它可以从初始化矩阵上下文时得到的rknn_matmul_io_attr结构体中获取。结构体定义如下表所示:

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| name | char[] | 矩阵的名字 |
| scale | float* | 矩阵的scale数组指针 |
| scale_len | int32_t | 矩阵的scale数组长度 |
| zp | int32_t* | 矩阵的zero_point数组指针 |
| zp_len | int32_t | 矩阵的zero_point数组长度 |

## rknn_matmul_shape

rknn_matmul_shape表示某个特定shape的矩阵乘法的M、K和N，在初始化动态shape的矩阵乘法上下文时，需要提供shape的数量，并使用rknn_matmul_shape结构体数组表示所有的输入的shape。结构体定义如下表所示:

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| M | int32_t | 矩阵A的行数 |
| K | int32_t | 矩阵A的列数 |
| N | int32_t | 矩阵B的列数 |

## 4.5 矩阵乘法API说明

### rknn_matmul_create

该函数的功能是根据传入的矩阵乘法规格等信息，完成矩阵乘法上下文的初始化，并返回输入和输出tensor的形状、大小和数据类型等信息。

| API | rknn_matmul_create |
| --- | --- |
| 功能 | 初始化矩阵乘法上下文。 |
| 参数 | rknn_matmul_ctx* ctx：矩阵乘法上下文指针。 |
| | rknn_matmul_info* info：矩阵乘法的规格信息结构体指针。 |
| | rknn_matmul_io_attr* io_attr：矩阵乘法输入和输出tensor属性结构体指针。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_matmul_info info;
memset(&info, 0, sizeof(rknn_matmul_info));
info.M            = 4;
info.K            = 64;
info.N            = 32;
info.type         = RKNN_INT8_MM_INT8_TO_INT32;
info.B_layout     = RKNN_MM_LAYOUT_NORM;
info.AC_layout    = RKNN_MM_LAYOUT_NORM;

rknn_matmul_io_attr io_attr;
memset(&io_attr, 0, sizeof(rknn_matmul_io_attr));

int ret = rknn_matmul_create(&ctx, &info, &io_attr);
if (ret < 0) {
    printf("rknn_matmul_create fail! ret=%d\n", ret);
    return -1;
}
```

## rknn_matmul_set_io_mem

该函数用于设置矩阵乘法运算的输入/输出内存。在调用该函数前，先使用rknn_create_mem接口创建的rknn_tensor_mem结构体指针，接着将其与rknn_matmul_create函数返回的矩阵A、B或C的rknn_matmul_tensor_attr结构体指针传入该函数，把输入和输出内存设置到矩阵乘法上下文中。在调用该函数前，要根据rknn_matmul_info中配置的内存排布准备好矩阵A和矩阵B的数据。

| API | rknn_matmul_set_io_mem |
| --- | --- |
| 功能 | 设置矩阵乘法的输入/输出内存。 |
| 参数 | rknn_matmul_ctx ctx：矩阵乘法上下文。 |
| | rknn_tensor_mem* mem：tensor内存信息结构体指针。 |
| | rknn_matmul_tensor_attr* attr：矩阵乘法输入和输出tensor属性结构体指针。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
// Create A
rknn_tensor_mem* A = rknn_create_mem(ctx, io_attr.A.size);
if (A == NULL) {
  printf("rknn_create_mem fail!\n");
  return -1;
}
memset(A->virt_addr, 1, A->size);
rknn_matmul_io_attr io_attr;
memset(&io_attr, 0, sizeof(rknn_matmul_io_attr));

int ret = rknn_matmul_create(&ctx, &info, &io_attr);
if (ret < 0) {
  printf("rknn_matmul_create fail! ret=%d\n", ret);
  return -1;
}
// Set A
ret = rknn_matmul_set_io_mem(ctx, A, &io_attr.A);
if (ret < 0) {
  printf("rknn_matmul_set_io_mem fail! ret=%d\n", ret);
  return -1;
}
```

## rknn_matmul_set_core_mask

该函数用于设置矩阵乘法运算时可用的NPU核心（仅支持RK3588和RK3576平台）。在调用该函数前，需要先通过rknn_matmul_create函数初始化矩阵乘法上下文。可通过该函数设置的掩码值，指定需要使用的核心，以提高矩阵乘法运算的性能和效率。

| API | rknn_matmul_set_core_mask |
| --- | --- |
| 功能 | 设置矩阵乘法运算的NPU核心掩码。 |
| 参数 | rknn_matmul_ctx ctx：矩阵乘法上下文。 |
| | rknn_core_mask core_mask：矩阵乘法运算的NPU核心掩码值，用于指定可用的NPU核心。掩码的每一位代表一个核心，如果对应位为1，则表示该核心可用；否则，表示该核心不可用（详细掩码说明见rknn_set_core_mask API参数）。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_matmul_set_core_mask(ctx, RKNN_NPU_CORE_AUTO);
```

## rknn_matmul_set_quant_params

rknn_matmul_set_quant_params用于设置每个矩阵的量化参数，支持Per-Channel量化、Per-Layer量化和Per-Group量化方式的量化参数设置。当使用Per-Group量化时，rknn_quant_params中的scale和zp数组的长度等于N*K/group_size。当使用Per-Channel量化时，rknn_quant_params中的scale和zp数组的长度等于N。当使用Per-Layer量化时，rknn_quant_params中的scale和zp数组的长度为1。在rknn_matmul_run之前调用此接口设置所有矩阵的量化参数。如果不调用此接口，则默认量化方式为Per-Layer量化，scale=1.0，zero_point=0。

| API | rknn_matmul_set_quant_params |
| --- | --- |
| 功能 | 设置矩阵的量化参数。 |
| 参数 | rknn_matmul_ctx ctx：矩阵乘法上下文。 |
| | rknn_quant_params* params： 矩阵的量化参数信息。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
rknn_quant_params params_a;
memcpy(params_a.name, io_attr.A.name, RKNN_MAX_NAME_LEN);
params_a.scale_len = 1;
params_a.scale = (float *)malloc(params_a.scale_len * sizeof(float));
params_a.scale[0] = 0.2;
params_a.zp_len = 1;
params_a.zp = (int32_t *)malloc(params_a.zp_len * sizeof(int32_t));
params_a.zp[0] = 0;
rknn_matmul_set_quant_params(ctx, &params_a);
```

## rknn_matmul_get_quant_params

rknn_matmul_get_quant_params用于rknn_matmul_type类型等于RKNN_INT8_MM_INT8_TO_INT32并且Per-Channel量化方式时，获取矩阵B所有通道scale归一化后的scale值，获取的scale值和A的原始scale值相乘可以得到C的scale值。可以用于在矩阵C没有真实scale时，近似计算得到C的scale。

| API | rknn_matmul_get_quant_params |
| --- | --- |
| 功能 | 获取矩阵B的量化参数。 |
| 参数 | rknn_matmul_ctx ctx：矩阵乘法上下文。 |
| | rknn_quant_params* params： 矩阵B的量化参数信息。 |
| | float* scale: 矩阵B的scale指针。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
float b_scale;
rknn_matmul_get_quant_params(ctx, &params_b, &b_scale);
```

## rknn_matmul_create_dyn_shape(deprecated)

该接口已废弃,改用rknn_matmul_create_dynamic_shape接口。

## rknn_matmul_create_dynamic_shape

rknn_matmul_create_dynamic_shape用于创建动态shape矩阵乘法上下文,该接口需要传入rknn_matmul_info结构体、shape数量以及对应的shape数组，shape数组会记录多个M、K和N值。在初始化成功后，会得到rknn_matmul_io_attr的数组，数组中包含了所有的输入输出矩阵的shape、大小和数据类型等信息。目前支持设置多个不同的M，K和N。

| API | rknn_matmul_create_dynamic_shape |
| --- | --- |
| 功能 | 初始化动态shape矩阵乘法的上下文。 |
| 参数 | rknn_matmul_ctx *ctx：矩阵乘法上下文指针。 |
| | rknn_matmul_info* info：矩阵乘法的规格信息结构体指针。其中M、K和N不需要设置。 |
| | int shape_num：矩阵上下文支持的shape数量。 |
| | rknn_matmul_shape dynamic_shapes[]：矩阵上下文支持的shape数组。 |
| | rknn_matmul_io_attr io_attrs[]：矩阵乘法输入和输出tensor属性结构体数组。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
const int           shape_num = 2;
rknn_matmul_shape shapes[shape_num];
for (int i = 0; i < shape_num; ++i) {
  shapes[i].M = i+1;
  shapes[i].K = 64;
  shapes[i].N = 32;
}
rknn_matmul_io_attr io_attr[shape_num];
memset(io_attr, 0, sizeof(rknn_matmul_io_attr) * shape_num);

int ret = rknn_matmul_create_dynamic_shape(&ctx, &info, shape_num, shapes, io_attr);
if (ret < 0) {
  fprintf(stderr, " rknn_matmul_create_dynamic_shape fail! ret=%d\n", ret);
  return -1;
}
```

### rknn_matmul_set_dynamic_shape

rknn_matmul_set_dynamic_shape用于指定矩阵乘法使用的某一个shape。在创建动态shape的矩阵乘法上下文后，选取其中一个rknn_matmul_shape结构体作为输入参数，调用此接口设置运算使用的shape。

| API | rknn_matmul_set_dynamic_shape |
| --- | --- |
| 功能 | 设置矩阵乘法shape。 |
| 参数 | rknn_matmul_ctx ctx：矩阵乘法上下文。 |
| | rknn_matmul_shape* shape：指定矩阵乘法使用的shape。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
ret = rknn_matmul_set_dynamic_shape(ctx, &shapes[0]);
if (ret != 0) {
  fprintf(stderr, "rknn_matmul_set_dynamic_shapes fail!\n");
  return -1;
}
```

## rknn_B_normal_layout_to_native_layout

rknn_B_normal_layout_to_native_layout用于将矩阵B的原始形状排列的数据（KxN）转换为高性能数据排列方式的数据。

| API | rknn_B_normal_layout_to_native_layout |
| --- | --- |
| 功能 | 将矩阵B的数据排列从原始形状转换成高性能形状。 |
| 参数 | void* B_input：原始形状的矩阵B数据指针。 |
| | void* B_output：高性能形状的矩阵B数据指针。 |
| | int K：矩阵B的行数。 |
| | int N：矩阵B的列数。 |
| | int subN：等于rknn_matmul_io_attr结构体中的B.dims[2]。 |
| | int subK：等于rknn_matmul_io_attr结构体中的B.dims[3]。 |
| | rknn_matmul_info* info：矩阵乘法的规格信息结构体指针。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
int32_t subN = io_attr.B.dims[2];
int32_t subK = io_attr.B.dims[3];
rknn_B_normal_layout_to_native_layout(B_Matrix, B->virt_addr, K, N, subN, subK, &info);
```

## rknn_matmul_run

该函数用于运行矩阵乘法运算，并将结果保存在输出矩阵C中。在调用该函数前，输入矩阵A和B需要先准备好数据，并通过rknn_matmul_set_io_mem函数设置到输入缓冲区。输出矩阵C需要先通过rknn_matmul_set_io_mem函数设置到输出缓冲区，而输出矩阵的tensor属性则通过rknn_matmul_create函数获取。

| API | rknn_matmul_run |
| --- | --- |
| 功能 | 运行矩阵乘法运算。 |
| 参数 | rknn_matmul_ctx ctx：矩阵乘法上下文。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
int ret = rknn_matmul_run(ctx);
```

## rknn_matmul_destroy

该函数用于销毁矩阵乘法运算上下文，释放相关资源。在使用完rknn_matmul_create函数创建的矩阵乘法上下文指针后，需要调用该函数进行销毁。

| API | rknn_matmul_destroy |
| --- | --- |
| 功能 | 销毁矩阵乘法运算上下文。 |
| 参数 | rknn_matmul_ctx ctx：矩阵乘法上下文。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
int ret = rknn_matmul_destroy(ctx);
```

## 4.6 自定义算子数据结构定义

### rknn_gpu_op_context

rknn_gpu_op_context表示指定GPU运行的自定义算子的上下文信息。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| cl_context | void* | OpenCL的cl_context对象，使用时请强制类型转换成cl_context。 |
| cl_command_queue | void* | OpenCL的cl_command_queue对象，使用时请强制类型转换成cl_command_queue。 |
| cl_kernel | void* | OpenCL的cl_kernel对象，使用时请强制类型转换成cl_kernel。 |

### rknn_custom_op_context

rknn_custom_op_context表示自定义算子的上下文信息。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| target | rknn_target_type | 执行自定义算子的后端设备：<br>RKNN_TARGET_TYPE_CPU： CPU<br>RKNN_TARGET_TYPE_CPU： GPU |
| internal_ctx | rknn_custom_op_interal_context | 算子内部的私有上下文。 |
| gpu_ctx | rknn_gpu_op_context | 包含自定义算子的OpenCL上下文信息，当执行后端设备是GPU时，在回调函数中从该结构体获取OpenCL的cl_context等对象。 |
| priv_data | void* | 留给开发者管理的数据指针。 |

### rknn_custom_op_tensor

rknn_custom_op_tensor表示自定义算子的输入/输出的tensor信息。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| attr | rknn_tensor_attr | 包含tensor的名称、形状、大小等信息。 |
| mem | rknn_tensor_mem | 包含tensor的内存地址、fd、有效数据偏移等信息。 |

### rknn_custom_op_attr

rknn_custom_op_attr表示自定义算子的参数或属性信息。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| name | char[] | 自定义算子的参数名。 |
| dtype | rknn_tensor_type | 每个元素的数据类型。 |
| n_elems | uint32_t | 元素数量。 |
| data | void* | 参数数据内存段的虚拟地址。 |

## rknn_custom_op

rknn_custom_op表示自定义算子的注册信息。结构体的定义如下表所示：

| 成员变量 | 数据类型 | 含义 |
| --- | --- | --- |
| version | uint32_t | 自定义算子版本号。 |
| target | rknn_target_type | 自定义算子执行后端类型。 |
| op_type | char[] | 自定义算子类型。 |
| cl_kernel_name | char[] | OpenCL的kernel函数名。 |
| cl_kernel_source | char* | OpenCL的资源名称。当cl_source_size等于0时，表示文件绝对路径；当cl_source_size大于0时，表示kernel函数代码的字符串。 |
| cl_source_size | uint64_t | 当cl_kernel_source是字符串，表示字符串长度；当cl_kernel_source是文件路径，则设置为0。 |
| cl_build_options | char[] | OpenCL kernel的编译选项。 |
| init | int (*)(rknn_custom_op_context* op_ctx, rknn_custom_op_tensor* inputs, uint32_t n_inputs, rknn_custom_op_tensor* outputs, uint32_t n_outputs); | 自定义算子初始化回调函数指针。在注册时，调用一次。不需要时可以设置为NULL。 |
| prepare | int (*)(rknn_custom_op_context* op_ctx, rknn_custom_op_tensor* inputs, uint32_t n_inputs, rknn_custom_op_tensor* outputs, uint32_t n_outputs); | 预处理回调函数指针。在rknn_run时调用一次。不需要时可以设置为NULL。 |
| compute | int (*)(rknn_custom_op_context* op_ctx, rknn_custom_op_tensor* inputs, uint32_t n_inputs, rknn_custom_op_tensor* outputs, uint32_t n_outputs); | 自定义算子功能的回调函数指针。在rknn_run时调用一次。**不能设置为NULL**。 |
| compute_native | int (*)(rknn_custom_op_context* op_ctx, rknn_custom_op_tensor* inputs, uint32_t n_inputs, rknn_custom_op_tensor* outputs, uint32_t n_outputs); | 高性能计算的回调函数指针，它与compute回调函数区别是输入和输出的tensor的格式存在差异。暂不支持，目前设置为NULL。 |
| destroy | int (*)(rknn_custom_op_context* op_ctx); | 销毁资源的回调函数指针。在rknn_destroy时调用一次。 |

## 4.7 自定义算子API说明

### rknn_register_custom_ops

在初始化上下文成功后，该函数用于在上下文中注册若干个自定义算子的信息，包括自定义算子类型、运行后端类型、OpenCL内核信息以及回调函数指针。注册成功后，在推理阶段，rknn_run接口会调用开发者实现的回调函数。

| API | rknn_register_custom_ops |
| --- | --- |
| 功能 | 注册若干个自定义算子到上下文中。 |
| 参数 | rknn_context *context：rknn_context指针。函数调用之前，context必须已经初始化成功。 |
| | rknn_custom_op* op：自定义算子信息数组，数组每个元素是rknn_custom_op结构体对象。 |
| | uint32_t custom_op_num：自定义算子信息数组长度。 |
| 返回值 | int 错误码（见RKNN返回值错误码）。 |

示例代码如下：

```
// CPU operators
rknn_custom_op user_op[2];
memset(user_op, 0, 2 * sizeof(rknn_custom_op));
strncpy(user_op[0].op_type, "cstSoftmax", RKNN_MAX_NAME_LEN - 1);
user_op[0].version = 1;
user_op[0].target  = RKNN_TARGET_TYPE_CPU;
user_op[0].init    = custom_op_init_callback;
user_op[0].compute = compute_custom_softmax_float32;
user_op[0].destroy = custom_op_destroy_callback;

strncpy(user_op[1].op_type, "ArgMax", RKNN_MAX_NAME_LEN - 1);
user_op[1].version = 1;
user_op[1].target  = RKNN_TARGET_TYPE_CPU;
user_op[1].init    = custom_op_init_callback;
user_op[1].compute = compute_custom_argmax_float32;
user_op[1].destroy = custom_op_destroy_callback;

ret = rknn_register_custom_ops(ctx, user_op, 2);
if (ret < 0) {
  printf("rknn_register_custom_ops fail! ret = %d\n", ret);
  return -1;
}
```

### rknn_custom_op_get_op_attr

该函数用于在自定义算子的回调函数中获取自定义算子的参数信息，例如Softmax算子的axis参数。它传入自定义算子参数的字段名称和一个rknn_custom_op_attr结构体指针，调用该接口后，参数值会存储在rknn_custom_op_attr结构体中的data成员中，开发者根据返回的结构体内dtype成员将该指针强制转换成C语言中特定数据类型的数组首地址，再按照元素数量读取出完整参数值。

| API | rknn_custom_op_get_op_attr |
| --- | --- |
| 功能 | 获取自定义算子的参数或属性。 |
| 参数 | rknn_custom_op_context* op_ctx：自定义算子上下文指针。 |
| | const char* attr_name：自定义算子参数的字段名称。 |
| | rknn_custom_op_attr* op_attr：表示自定义算子参数值的结构体。 |
| 返回值 | 无 |

示例代码如下：

```
rknn_custom_op_attr op_attr;
rknn_custom_op_get_op_attr(op_ctx, "axis", &op_attr);
if (op_attr.n_elems == 1 && op_attr.dtype == RKNN_TENSOR_INT64) {
    axis = ((int64_t*)op_attr.data)[0];
}
…
```

# 5. RKNN返回值错误码

RKNN API函数的返回值错误码定义如下表所示：

| 错误码 | 错误详情 |
| --- | --- |
| RKNN_SUCC(0) | 执行成功。 |
| RKNN_ERR_FAIL(-1) | 执行出错。 |
| RKNN_ERR_TIMEOUT(-2) | 执行超时。 |
| RKNN_ERR_DEVICE_UNAVAILABLE(-3) | NPU设备不可用。 |
| RKNN_ERR_MALLOC_FAIL(-4) | 内存分配失败。 |
| RKNN_ERR_PARAM_INVALID(-5) | 传入参数错误。 |
| RKNN_ERR_MODEL_INVALID(-6) | 传入的RKNN模型无效。 |
| RKNN_ERR_CTX_INVALID(-7) | 传入的rknn_context无效。 |
| RKNN_ERR_INPUT_INVALID(-8) | 传入的rknn_input对象无效。 |
| RKNN_ERR_OUTPUT_INVALID(-9) | 传入的rknn_output对象无效。 |
| RKNN_ERR_DEVICE_UNMATCH(-10) | 版本不匹配。 |
| RKNN_ERR_INCOMPATILE_OPTIMIZATION_LEVEL_VERSION(-12) | RKNN模型设置了优化等级的选项，但是和当前驱动不兼容。 |
| RKNN_ERR_TARGET_PLATFORM_UNMATCH(-13) | RKNN模型和当前平台不兼容。 |